Package: descem (via r-universe)

October 9, 2024

Title Discrete Event Simulation for Cost-Effectiveness Modelling

Version 0.1.2

Description A package designed to perform discrete event simulation for cost-effectiveness modelling.

License Apache License (>= 2)

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

Suggests flexsurv, dplyr, ggplot2, knitr, rmarkdown, kableExtra, DiagrammeR, MASS

Imports purrr, tidyr, doParallel, data.table, foreach, stats, utils

VignetteBuilder knitr

Repository https://pharmaverse.r-universe.dev

RemoteUrl https://github.com/Roche/descem

RemoteRef HEAD

RemoteSha 12da8ab494e80f47628f8080761b06de76348d4d

Contents

`add_cost` *Defining costs for events and intervention*

Description

Defining costs for events and intervention

Usage

```
add_cost(.data = NULL, cost, evt, trt, cycle_l = NULL, cycle_starttime = 0)
```

Arguments

`.data`	Existing cost data
`cost`	Value or expression to calculate the cost estimate
`evt`	Vector of events for which this cost is applicable
`trt`	Vector of interventions for which this cost is applicable
`cycle_l`	Cycle length; only needed if costs are calculated per cycle
`cycle_starttime`	Cycle when costs start being accrued; only needed if costs are calculated per cycle

Details

Costs can be defined by writing expressions and objects in the cost argument whose execution will be delayed until the model runs.

This function accepts the use of pipes (%>%) to define multiple costs.

Value

A list of costs

Examples

```
add_cost(evt = c("start","idfs","ttot"),trt = "int",cost = cost.int*fl.int + cost.idfs)
```

add_item *Defining parameters that may be used in model calculations*

Description

Defining parameters that may be used in model calculations

Usage

```
add_item(.data = NULL, ...)
```

Arguments

.data	Existing data
...	Items to define for the simulation

Details

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `add_item` with a list of one element, it's better to group them into a single `add_item` with a list of two elements.

Value

A list of items

Examples

```
add_item(fl.idfs = 0)
add_item(util_idfs = if(psa_bool){rnorm(1,0.8,0.2)} else{0.8}, util.mbc = 0.6, cost_idfs = 2500)
```

add_reactevt *Define the modifications to other events, costs, utilities, or other items affected by the occurrence of the event*

Description

Define the modifications to other events, costs, utilities, or other items affected by the occurrence of the event

Usage

```
add_reactevt(.data = NULL, name_evt, input)
```

Arguments

.data	Existing data for event reactions
name_evt	Name of the event for which reactions are defined.
input	Expressions that define what happens at the event, using functions as defined in the Details section

Details

There are a series of objects that can be used in this context to help define the event reactions.

The following functions may be used to define event reactions within this add_reactevt() function: modify_item() | Adds & Modifies items/flags/variables for future events new_event() | Adds events to the vector of events for that patient modify_event() | Modifies existing events by changing their time

Apart from the items defined with add_item(), we can also use standard variables that are always defined within the simulation: curtime | Current event time (numeric) prevtime | Time of the previous event (numeric) cur_evtlist | Named vector of events that is yet to happen for that patient (named numeric vector) evt | Current event being processed (character) i | Patient being iterated (character) simulation | Simulation being iterated (numeric)

The model will run until curtime is set to Inf, so the event that terminates the model should modify curtime and set it to Inf.

Examples

```
add_reactevt(name_evt = "start",input = {})
add_reactevt(name_evt = "idfs",input = {modify_item(list("fl.idfs"= 0))})
```

---

add_tte *Define events and the initial event time*

Description

Define events and the initial event time

Usage

```
add_tte(.data = NULL, trt, evts, other_inp = NULL, input)
```

Arguments

.data	Existing data for initial event times
trt	The intervention for which the events and initial event times are defined
evts	A vector of the names of the events
other_inp	A vector of other input variables that should be saved during the simulation
input	The definition of initial event times for the events listed in the evts argument

Details

Events need to be separately defined for each intervention.

For each event that is defined in this list, the user needs to add a reaction to the event using the `add_reactevt()` function which will determine what calculations will happen at an event.

Value

A list of initial events and event times

Examples

```
add_tte(trt="int",evts = c("start","ttot","idfs","os"),
input={
start <- 0
idfs <- draw_tte(1,'lnorm',coef1=2, coef2=0.5)
ttot <- min(draw_tte(1,'lnorm',coef1=1, coef2=4),idfs)
os <- draw_tte(1,'lnorm',coef1=0.8, coef2=0.2)
})
```

---

add_util *Defining utilities for events and interventions*

Description

Defining utilities for events and interventions

Usage

```
add_util(.data = NULL, util, evt, trt, cycle_l = NULL, cycle_starttime = 0)
```

Arguments

`.data`	Existing utility data
`util`	Value or expression to calculate the utility estimate
`evt`	Events for which this utility is applicable
`trt`	Interventions for which this utility is applicable
`cycle_l`	Cycle length; only needed if utilities are calculated per cycle
`cycle_starttime`	Cycle when utilities start being accrued; only needed if utilities are calculated per cycle

Details

Utilities can be defined by writing expressions and objects in the cost argument whose execution will be delayed until the model runs.

This function accepts the use of pipes (%>%) to define multiple utilities.

Value

A list of utilities

Examples

```
add_util(evt = c("start","idfs","ttot"),
trt = c("int", "noint"),
util = util.idfs.ontx * fl.idfs.ontx + util.idfs.offtx * (1-fl.idfs.ontx))
```

`ceac_des` *Calculate the cost-effectiveness acceptability curve (CEAC) for a DES model with a PSA result*

Description

Calculate the cost-effectiveness acceptability curve (CEAC) for a DES model with a PSA result

Usage

```
ceac_des(wtp, results, interventions = NULL)
```

Arguments

`wtp`	Vector of length >=1 with the willingness to pay
`results`	The list object returned by `RunSim()`
`interventions`	A character vector with the names of the interventions to be used for the analysis

Value

A data frame with the CEAC results

Examples

```
## Not run:
ceac_des(seq(from=10000,to=500000,by=10000),results)

## End(Not run)
```

draw_beta *Draw from a beta distribution based on mean and se*

Description

Draw from a beta distribution based on mean and se

Usage

```
draw_beta(value, se, seed = NULL)
```

Arguments

value	A vector of the mean values
se	A vector of the standard errors of the means
seed	An integer which will be used to set the seed for this draw.

Value

A single estimate from the beta distribution based on given parameters

Examples

```
draw_beta(value=0.8,se=0.2)
```

draw_gamma *Draw from a gamma distribution based on mean and se*

Description

Draw from a gamma distribution based on mean and se

Usage

```
draw_gamma(value, se, seed = NULL)
```

Arguments

value	A vector of the mean values
se	A vector of the standard errors of the means
seed	An integer which will be used to set the seed for this draw.

Value

A single estimate from the gamma distribution based on given parameters

Examples

```
draw_gamma(value=0.8,se=0.2)
```

`draw_resgompertz` *Draw from a restricted Gompertz distribution*

Description

Draw from a restricted Gompertz distribution

Usage

```
draw_resgompertz(
  n,
  shape,
  rate,
  lower_bound = 0,
  upper_bound = Inf,
  seed = NULL
)
```

Arguments

`n`	The number of observations to be drawn
`shape`	The shape parameter of the Gompertz distribution, defined as in the coef() output on a flexsurvreg object
`rate`	The rate parameter of the Gompertz distribution, defined as in the coef() output on a flexsurvreg object
`lower_bound`	The lower bound of the restricted distribution
`upper_bound`	The upper bound of the restricted distribution
`seed`	An integer which will be used to set the seed for this draw.

Value

Estimate(s) from the restricted Gompertz distribution based on given parameters

Examples

```
draw_resgompertz(1,shape=0.05,rate=0.01,lower_bound = 50)
```

draw_tte *Draw a time to event from a list of parametric survival functions*

Description

Draw a time to event from a list of parametric survival functions

Usage

```
draw_tte(
  n_chosen = 1,
  dist = "exp",
  coef1 = 1,
  coef2 = NULL,
  coef3 = NULL,
  hr = 1,
  seed = NULL
)
```

Arguments

n_chosen	The number of observations to be drawn
dist	The distribution; takes values 'lnorm','weibullPH','weibull','llogis','gompertz','gengamma','gamma','ex
coef1	First coefficient of the distribution, defined as in the coef() output on a flexsurvreg object
coef2	Second coefficient of the distribution, defined as in the coef() output on a flexsurvreg object
coef3	Third coefficient of the distribution, defined as in the coef() output on a flexsurvreg object
hr	A hazard ratio
seed	An integer which will be used to set the seed for this draw.

Value

A vector of time to event estimates from the given parameters

Examples

```
draw_tte(n_chosen=1,dist='exp',coef1=1,hr=1)
```

evpi_des — *Calculate the Expected Value of Perfect Information (EVPI) for a DES model with a PSA result*

Description

Calculate the Expected Value of Perfect Information (EVPI) for a DES model with a PSA result

Usage

```
evpi_des(wtp, results, interventions = NULL)
```

Arguments

wtp	Vector of length >=1 with the willingness to pay
results	The list object returned by RunSim()
interventions	A character vector with the names of the interventions to be used for the analysis

Value

A data frame with the EVPI results

Examples

```
## Not run:
evpi_des(seq(from=10000,to=500000,by=10000),results)

## End(Not run)
```

extract_psa_result — *Extract PSA results from a treatment*

Description

Extract PSA results from a treatment

Usage

```
extract_psa_result(x, element, trt)
```

Arguments

x	The output_psa data frame from the list object returned by RunSim()
element	Variable for which PSA results are being extracted (single string)
trt	Intervention for which PSA results are being extracted (single string)

Value

A dataframe with PSA results from the specified intervention

Examples

```
## Not run:
extract_psa_result(results$output_psa,"costs","int")

## End(Not run)
```

`modify_event` *Modify the time of existing events*

Description

Modify the time of existing events

Usage

```
modify_event(evt, env_ch = NULL)
```

Arguments

`evt`	A list of events and their times
`env_ch`	Environment in which to save list (should not be defined by user)

Details

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `modify_event` with a list of one element, it's better to group them into a single `modify_event` with a list of two elements.

Examples

```
## Not run:
modify_event(list("os"=40, "ttot"=curtime+0.0001))

## End(Not run)
```

modify_item *Modify the value of existing items*

Description

Modify the value of existing items

Usage

```
modify_item(list_item, env_ch = NULL)
```

Arguments

list_item	A list of items and their values or expressions
env_ch	Environment in which to save list (should not be defined by user)

Details

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `modify_item` with a list of one element, it's better to group them into a single `modify_item` with a list of two elements.

Examples

```
## Not run:
modify_item(list(cost.idfs = 500, cost.tx = cost.tx + 4000))

## End(Not run)
```

new_event *Generate new events to be added to existing vector of events*

Description

Generate new events to be added to existing vector of events

Usage

```
new_event(evt, env_ch = NULL)
```

Arguments

evt	Event name and event time
env_ch	Environment in which to save list (should not be defined by user)

Details

The functions to add/modify events/inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `new_event` with a list of one element, it's better to group them into a single `new_event` with a list of two elements.

Examples

```
## Not run:
new_event(list("ae"=5))
new_event(list("ae"=5,"nat.death" = 100))

## End(Not run)
```

`RunSim` *Run the simulation*

Description

Run the simulation

Usage

```
RunSim(
  trt_list = c("int", "noint"),
  common_all_inputs = NULL,
  common_pt_inputs = NULL,
  unique_pt_inputs = NULL,
  init_event_list = NULL,
  evt_react_list = evt_react_list,
  util_ongoing_list = NULL,
  util_instant_list = NULL,
  util_cycle_list = NULL,
  cost_ongoing_list = NULL,
  cost_instant_list = NULL,
  cost_cycle_list = NULL,
  npats = 500,
  n_sim = 1,
  psa_bool = NULL,
  ncores = 1,
  drc = 0.035,
  drq = 0.035,
  input_out = NULL,
  ipd = TRUE,
  debug = FALSE
)
```

Arguments

trt_list	A vector of the names of the interventions evaluated in the simulation
common_all_inputs	A list of inputs common across patients that do not change within a simulation
common_pt_inputs	A list of inputs that change across patients but are not affected by the intervention
unique_pt_inputs	A list of inputs that change across each intervention
init_event_list	A list of initial events and event times. If no initial events are given, a "Start" event at time 0 is created automatically
evt_react_list	A list of event reactions
util_ongoing_list	A list of utilities that are accrued at an ongoing basis
util_instant_list	A list of utilities that are accrued instantaneously at an event
util_cycle_list	A list of utilities that are accrued in cycles
cost_ongoing_list	A list of costs that are accrued at an ongoing basis
cost_instant_list	A list of costs that are accrued instantaneously at an event
cost_cycle_list	A list of costs that are accrued in cycles
npats	The number of patients to be simulated
n_sim	The number of simulations to run per patient
psa_bool	A boolean to determine if PSA should be conducted. If n_sim > 1 and psa_bool = FALSE, the differences between simulations will be due to sampling
ncores	The number of cores to use for parallel computing
drc	The discount rate for costs
drq	The discount rate for LYs/QALYs
input_out	A vector of variables to be returned in the output data frame
ipd	A boolean to determine if individual patient data should be returned. If set to false, only the main aggregated outputs will be returned (slightly speeds up code)
debug	A boolean to determine if non-parallel RunEngine function should be used, which facilitates debugging. Setting this option to true will ignore the value of ncores

Value

A list of data frames with the simulation results

Examples

```
## Not run:
RunSim(trt_list=c("int","noint"),
common_all_inputs = common_all_inputs,
common_pt_inputs = common_pt_inputs,
unique_pt_inputs = unique_pt_inputs,
init_event_list = init_event_list,
evt_react_list = evt_react_list,
util_ongoing_list = util_ongoing_list,
util_instant_list = util_instant_list,
cost_ongoing_list = cost_ongoing_list,
cost_instant_list = cost_instant_list,
npats = 500,
n_sim = 1,
psa_bool = FALSE,
ncores = 1,
drc = 0.035,
drq = 0.035,
ipd = TRUE)

## End(Not run)
```

---

summary_results_det *Deterministic results for a specific treatment*

Description

Deterministic results for a specific treatment

Usage

```
summary_results_det(out = final_output, trt = NULL)
```

Arguments

out	The final_output data frame from the list object returned by RunSim()
trt	The reference treatment for calculation of incremental outcomes

Value

A dataframe with absolute costs, LYs, QALYs, and ICER and ICUR for each intervention

Examples

```
## Not run:
summary_results_det(results$final_output,trt="int")

## End(Not run)
```

summary_results_psa *Summary of PSA outputs for a treatment*

Description

Summary of PSA outputs for a treatment

Usage

```
summary_results_psa(out = output_psa, trt = NULL)
```

Arguments

out	The output_psa data frame from the list object returned by RunSim()
trt	The reference treatment for calculation of incremental outcomes

Value

A data frame with mean and 95% CI of absolute costs, LYs, QALYs, ICER and ICUR for each intervention from the PSA samples

Examples

```
## Not run:
summary_results_psa(results$output_psa, trt="int")

## End(Not run)
```

Index